CSc 174 Database Management Systems

15. Database Security

Ying Jin
Computer Science Department
California state University, Sacramento

Threats to Databases

- Loss of integrity
  - Protect information from improper modification
    - Creation, insertion, modification, deletion
- Loss of availability
  - Data items should be available to legitimate users
- Loss of confidentiality
  - Protection from unauthorized disclosure

Control Measures Against Threats

- Access control
  - Discretionary security mechanisms
  - Mandatory security mechanisms
  - Role-based access control

Discretionary Access Control

- Based on Granting and revoking of privileges
- Query statements to allow the DBA to grant and revoke privileges
- Many current relational DBMSs use some variation of this technique

Types of Discretionary Privileges

- The account level
  - DBA specifies privileges to account
  - Independently of the relations in the database
- The relational (or table) level
  - DBA control the privilege to access each individual relation or view

Problem with Discretionary Authorization

- Do not impose any control on how information is propagated and used once it has been accessed by uses authorized to do so.

Mandatory Access Control

- Classify data and users based on security classes
- Most commercial DBMSs currently only support discretionary access control
- Mandatory access control
  - Government, military, intelligence applications, industrial applications

Typical Security Classes

- Top secret (TS)
- Secret (S)
- Confidential (C)
- Unclassified (U)
- TS>=S>=C>=U
- To rigid to require a strict classification of subjects and objects into security levels

Role-Based Access Control

- Managing security in large-scale enterprise-wide systems
- Permissions/privilege are associated with roles
- Users are assigned to appropriate roles
- GRANT, REVOKE privileges from roles
- Viable alternative to discretionary and mandatory access controls

RBAC (Cont.)

- Organize roles to reflect organization's lines of authority and responsibility
- Junior roles at the bottom are connected to progressively senior roles as one moves up the hierarchy
- Roles can be assigned temporal constraints

SQL Injection Attacks

What is SQL Injection Attacks

- Input data is inserted into a query
- Crafted malicious input can cause the query processor to misinterpret the intended query.
- Example (E1)

Warnings

- “Do not try any of these examples on any Web applications or systems, unless you have permission (in writing, preferably) from the application or system owner. In the the United States, you could be prosecuted under the Computer Fraud and Abuse Act of 1986 or the USA PATRIOT Act of 2001.” [1]

SQL Injection Attacks

- First order Injection
  - conduct attacks against the target system immediately after the injected malicious text arrives at the system
  - Tautology-based SQL Injection
  - Statement Injection
- Second order

Tautology-based SQL Injection

- A SQL tautology: a statement that is always true
- inserting a tautology into a conditional statement
- Example (E2)
- Patterns
  - 1=1
  - “<”, “>=”, “between”, “in”
  - Example (E3)

Tautology-based SQL Injection-E2

```
ResultSet rs1=null;
String sql="select * from users where loginname='"+name+"' and
    pwd='"+pass+"'";
System.out.println("sql:"+sql);
rs1=stmt1.executeQuery(sql);

 if (rs1.next())
    {
do {
System.out.println("id:"+rs1.getString(1)+"\t"+"loginname:"+rs1.getString(2)
    +"\t"+"pwd:"+rs1.getString(3)+"\t"+"email:"+rs1.getString(4));
         } while (rs1.next());
     }
else
     System.out.println("Login failed.");
```

Loginname: ’or 1=1 - -

Statement Injection

- insert additional statements into the original SQL statement
- Union statement
  - Example (E4)
- Piggybacked query
  - ; statement
  - Oracle does not allow ";" in a single PL/SQL statement.
  - Oracle allows it in PL/SQL blocks (E5)

Statement Injection – E4

- Enter Login name: ` union select * from employee –
- Select *

  from users

  where loginname = ` ′

  union select * from employee -- ` and pwd = ‘anypwd’

Piggybacked query -Oracle procedure Example (E5)

```
CREATE OR REPLACE PROCEDURE update_emp (pid IN
    varchar2, eid IN varchar2)
IS
    stmt varchar2(4000);
BEGIN
    stmt:='begin
          update employee set position_id=''' || pid || '''
    where id= '''|| eid || ''';' || 'END;';
    DBMS_OUTPUT.PUT_LINE('stmt: ' || stmt);
    EXECUTE IMMEDIATE stmt;
END;
/
```

Piggybacked query -Oracle procedure E5 (Cont.)

- Enter employee Id:

  111' ; DELETE FROM orders WHERE oid='A0001
- Update employee set position_id = ‘8888’ where id = '111' ; DELETE FROM orders WHERE oid = 'A0001’
- Other statements: DROP TABLE, UPDATE, INSERT, and SHUTDOWN

Injection to Store Procedures

- Vulnerable: when use dynamic SQL (e.g. E5)
- Static SQL
  - compile-time-fixed text
  - Should be used for security purpose
- Dynamic SQL
  - full text of is unknown at compile time
  - More flexible

Second Order Injection

- The attack and the SQL injection do not associate with each other in real time.
- The malicious user inputs are first stored in a system and then wait for being used by an application in the future

Second Order Injection – E6

```
String insertsql = "insert into users values (?,?,?,?)";
PreparedStatement pstmt =
    conn.prepareStatement(insertsql);
pstmt.setString(1,"567");
pstmt.setString(2,"admin'--");
pstmt.setString(3,"1234");
pstmt.setString(4,"fake@email.com");
pstmt.executeUpdate();
pstmt.close();
conn.close();
```

Second Order Injection – E6(cont)

- **Later:** use the update statement to change admin password:
- sql = “update users set password = ‘ ” + new_password + "’ where username = ‘ " + username + “ ‘ and password =’ “ + old_password + " ’ "

- Input Username: admin ‘ --
- Input Password: anypassword
- update users set password = ‘new_pass’ where username = ‘ admin ‘ -- ‘ and password = ‘old_pass’
- Alter the admin’s password to the attackers’ password

Fingerprinting and Enumeration

- Getting knowledge about the target system
- E.g. database type and structure, table schemas, user privilege levels
- Submit some specifically crafted user inputs
- learn from the error messages returned by the target system

SQL Denial of Service Attacks

- Overload a target system
- Submitting queries that would consume a large amount of system sources

Building Secured Applications

- How to build a new secured application
  - Defensive programming
- How to protect an existing system
  - Research Projects

Defensive Programming Techniques for coding an application

Validating Input

- Why blacklisting does not work?
- Use whitelist input validation
  - Test a given input is within a well-defined set of values that are known to be safe
- Validate various factors such as type, size, range

Consider Output

- Attackers want to know DB schema
  - Use Error Message (see Fingerprinting and Enumeration)
- Do not display configuration errors and exception handling
- Replace with general errors

Encrypting Sensitive Data

- Encrypt SSN, credit card number
- Attackers need to obtain the key to compromise the data
- How to design and implement is not a trivial task
    - Where to store the key?

Privilege Management

- Apply the principle of least privilege to user accounts
  - Can only query restricted tables, views
  - May not run arbitrary insert, update, or delete statement
- Revoking any unnecessary privileges of user

Static or dynamic SQL

- Use static SQL when possible
- When dynamic SQL is needed, try to use prepared statements if possible
  - Example of using prepared statement (E7)
  - Cannot solve all the problem (E8)

Prepare Statement Example (E7)

```
String sql="select * from users where loginname=? and pwd=?";
System.out.println("sql:"+sql);
PreparedStatement pstmt1 = conn.prepareStatement(sql);
pstmt1.setString(1,name);
pstmt1.setString(2,pass);
ResultSet rs1=null;
rs1=pstmt1.executeQuery();
```

- Handle E1, E2,E3,E4
- Consider each parameter as a input data (not control)

Prepared statement problem (E8)

- E6 is an example of injection when use prepared statement
- If use statement, then the injection is not allowed.

```
String insertsql = "insert into users values
    ('567','admin'--','1234','fake@email.com')";
stmt.executeUpdate(insertsql);
```

- ORA-00917: missing comma

Research Projects

- Parsed tree approach
- Non-Deterministic Finite Automaton
- Parameterized views

Tools to use

Many tools available to check different types of vulnerabilities in a web application.

Books referenced:

[1] Justin Clarke, *SQL Injection Attacks and Defense*, Syngress publishing, 2009

[2] Neil Daswani, Christoph Kern, and Anita Kesavan, *Foundations of Security: what every programmer needs to know*, Apress publishing, 2007

[3] R. Elmaseri and S. Navathe, *Fundamentals of Database System*, 7th Edition, Addison-Wesley.